MSFragger Manual

(build 20170103.0)

Introduction

MSFragger is an ultrafast database search tool for peptide identifications in mass spectrometry-based proteomics. It differs from conventional search engines by computing similarity scores in a fragment-centric fashion using a theoretical fragment index of candidate peptides. The speed of MSFragger makes it particularly suitable for 'open' database searches, where the precursor mass tolerance is set to hundreds of Daltons, for the identification of modified peptides. MSFragger is implemented in the cross-platform Java programming language and is compatible with standard proteomics file formats such as MGF/mzXML/mzML/pepXML.

Equipment

Computer Hardware requirements

The processor requirements of MSFragger depends on the complexity of your search (and your patience to wait for search results). For an open search (500Da precursor mass window) using a tryptic digest of the human proteome, a single processor core can search roughly 40,000 MS/MS spectra in under an hour. MSFragger scales well with the number of processor cores and runtimes of under 2 minutes per file have been achieved using a 28-core workstation. A desktop workstation with a quad core processor is sufficient for most simple workflows.

MSFragger requires substantial amounts of memory due to its in-memory fragment index. While MSFragger can operate with less memory than needed to store the fragment index, it will cause index fragmentation where it breaks the search into multiple passes, searching each input file against a small segment of the index at a time (which greatly increases the runtime). For the human Uniprot protein database with reversed decoys, approximately 3700 MB of memory is needed to prevent index fragmentation. The actual size of the fragment index is substantially lower (MSFragger uses a very conservative estimate of the available free memory to avoid out of memory situations). Specifying common modifications may boost memory requirements to 6 GB. Semi-tryptic, non-enzymatic, and phospho searches may take tens of gigabytes to avoid fragmented searches. Limiting the range of peptide lengths can reduce the search space and reduce memory consumption in such cases. While fragment index fragmentation is undesirable, it may be unavoidable in certain instances.

We recommend at least 8GB of memory for workflows involving standard tryptic digestions.

Operating System requirements

MSFragger has been tested on Mac OS X, Windows 7, and a number of Linux distributions. Note that a 64-bit operating system is required to access more than 4GB of memory.

Java requirements

MSFragger is written using Java 1.8 and requires the Java 8 Runtime Environment. We recommend the Oracle Java 8 Runtime (download and installation instructions are available at www.java.com).

Procedure

Preparing Input Files

Mass spectrometry data must first be converted to one of the supported MS/MS input formats of MGF, mzXML, or mzML. A popular option for converting from vendor file inputs and between various input formats is Proteowizard (proteowizard.sourceforge.net). MSFragger determines the appropriate data parser to use based on the file extension (.mgf for MGF, .mzXML for mzXML, and .mzML for mzML) and does not make inferences from file contents (i.e. naming a mzML file with the .mzXML extension will lead to unpredictable results or crashes).

The protein database must be supplied in FASTA format. MSFragger does not have the capability to generate decoys internally so they must be generated externally and appended to the protein database before running MSFragger.

Configuring MSFragger

Extract the MSFragger.jar into your working directory along with the sample configuration file called fragger.params. MSFragger is configured using a text parameters file. The parameters file is passed as the first argument to MSFragger and has no restrictions on names or file extensions (so one might want to name their configuration files to be more descriptive such as Uniprot_open_withmods.txt) after editing the parameters file for a particular analysis.

Parameter names are given left of the equal sign and parameter values are given to the right (e.g. num_threads = 4). White spaces are trimmed from the ends of each value by MSFragger. All text to the right of (and including) the # sign of each line is discarded so # can be used for comments in the parameters file.

General Parameters

num_threads	Number of CPU threads to use, should be set to the number of logical processors; a value of 0 (auto-detect) will cause MSFragger to use the auto-detected number of processors Default: 0
database_name	Path to the protein database file in FASTA format

Search Tolerances

precursor_mass_tolerance	Precursor mass tolerance (window is +/- this value) Default: 20
precursor_mass_units	Precursor mass tolerance units (0 for Da, 1 for ppm) Default: 1
precursor_true_tolerance	True precursor mass tolerance (window is +/- this value). Used for tie breaker of results (in spectrally ambiguous cases) and zero bin boosting in open searches (0 disables these features). This option is STRONGLY recommended for open searches. Default: 0
precursor_true_units	True precursor mass tolerance units (0 for Da, 1 for ppm)

	Default: 1
fragment_mass_tolerance	Fragment mass tolerance (window is +/- this value) Default: 20
fragment_mass_units	Fragment mass tolerance units (0 for Da, 1 for ppm) Default: 1
isotope_error	Isotope correction for MS/MS events triggered on isotopic peaks. Should be set to 0 (disabled) for open search or 0/1/2 for correction of narrow window searches. Shifts the precursor mass window to multiples of this value multiplied by the mass of C13-C12. Default: 0

In-silico Digestion Parameters

search_enzyme_name	Name of enzyme to be written to the pepXML file. Default: Trypsin
search_enzyme_cutafter	Residues after which the enzyme cuts (specified as a string of amino acids) Default: KR
search_enzyme_butnotafter	Residues that the enzyme will not cut before (misnomer: should really be called butnotbefore) Default: P
num_enzyme_termini	Number of enzyme termini (0, 1, or 2 for non-enzymatic, semi-enzymatic, fully-enzymatic) Default: 2
allowed_missed_cleavage	Allowed number of missed cleavages Default: 2

digest_min_length	Minimum length of peptides to be generated during in-silico digestion Default: 7
digest_max_length	Maximum length of peptides to be generated during in-silico digestion Default: 64
digest_mass_range	Mass range of peptides to be generated during in-silico digestion in Daltons (specified as a space separated range) Default: 500.0 5000.0

Variable Modification Parameters

clip_nTerm_M	Specifies the trimming of a protein N-terminal methionine as a variable modification (0 or 1) Default: 0
variable_mod_01 .. 07	Sets variable modifications. (variable_mod_01 to variable_mod_07). Space separated values with 1st value being the modification mass and the second being the residues (specified consecutively as a string) it modifies. * is used to represent any amino acid [is a modifier for protein N-terminal] is a modifier for protein C-terminal n is a modifier for peptide N-terminal c is a modifier for peptide C-terminal Syntax Examples: 15.9949 M (for oxidation on methionine) 79.66331 STY (for phosphorylation) -17.0265 nQnC (for pyro-Glu or loss of ammonia at peptide N-terminal) Example (M oxidation and N-terminal acetylation):

	variable_mod_01 = 15.9949 M
	variable_mod_02 = 42.0106 [*

allow_multiple_variable_mods_on_residue	Allow each amino acid to be modified by multiple variable modifications (0 or 1) Default: 1
max_variable_mods_per_mod	Maximum number of residues that can be occupied by each variable modification (maximum of 5). Default: 2
max_variable_mods_combinations	Maximum allowed number of modified variably modified peptides from each peptide sequence, (maximum of 65534). If a greater number than the maximum is generated, only the unmodified peptide is considered. Default: 5000

Spectrum Processing Parameters

minimum_peaks	Minimum number of peaks in experimental spectrum for matching Default: 10
use_topN_peaks	Pre-process experimental spectrum to only use top N peaks Default: 50
minimum_ratio	Filters out all peaks in experimental spectrum less intense than this multiple of the base peak intensity Default: 0.0
clear_mz_range	Removes peaks in this m/z range prior to matching. Useful for iTRAQ/TMT experiments (i.e. 0.0 150.0). Default: 0.0 0.0
max_fragment_charge	Maximum charge state for theoretical fragments to

	match (1-4). Default: 2
override_charge	Ignores precursor charge and uses charge state specified in precursor_charge range (0 or 1) Default: 0
precursor_charge	Assume range of potential precursor charge states. Only relevant when override_charge is set to 1. Specified as space separated range of integers. Default: 1 4

Open Search Features

track_zero_topN	Track top N unmodified peptide results separately from main results internally for boosting features. Should be set to a number greater than output_report_topN if zero bin boosting is desired. Default: 0
zero_bin_accept_expect	Ranks a zero-bin hit above all non-zero-bin hit if it has expectation less than this value. Default: 0.0
zero_bin_mult_expect	Multiplies expect value of PSMs in the zero-bin during results ordering (set to less than 1 for boosting). Default: 1.0
add_topN_complementary	Inserts complementary ions corresponding to the top N most intense fragments in each experimental spectra. Useful for recovery of modified peptides near C-terminal in open search. Should be set to 0 (disabled) otherwise. Default: 0

Modeling and Output Parameters

min_fragments_modelling	Minimum number of matched peaks in PSM for inclusion in statistical modeling Default: 3
min_matched_fragments	Minimum number of matched peaks for PSM to be reported. We recommend a minimum of 4 for narrow window searching and 6 for open searches. Default: 4
output_file_extension	File extension of output files Default: pep.xml
output_format	File format of output files (pepXML or tsv) Default: pepXML
output_report_topN	Reports top N PSMs per input spectrum Default: 1
output_max_expect	Suppresses reporting of PSM if top hit has expectation greater than this threshold Default: 50.0

Static Modification Parameters

add_Cterm_peptide	Statically add mass in Da to C-terminal of peptide Default: 0.0
add_Nterm_peptide	Statically add mass in Da to N-terminal of peptide Default: 0.0
add_Cterm_protein	Statically add mass in Da to C-terminal of protein Default: 0.0
add_Nterm_protein	Statically add mass in Da to N-terminal of protein

	Default: 0.0
add_C_cysteine ... add_X_usertext	Statically add mass to cysteine (or whatever amino acid is specified after ‘add_’). Examples: add_C_cysteine = 57.021464 add_K_lysine = 144.1021 Default: 0.0

Running MSFragger

Performance Considerations for Batch Processing

MSFragger allows multiple MS/MS input files to be processed in a batch. Passing multiple files to MSFragger at once allows MSFragger to reuse the fragment index for subsequent MS/MS run. This is particularly important for narrow window searches which may only take fractions of a second.

On computers or compute clusters with many processor cores, we highly recommended that MSFragger is set to process files sequentially with all available processor cores rather than running multiple instances of MSFragger in parallel (assigning a smaller number of cores to each). This reduces initialization times and allows the fragment index to be re-used, at the same time reducing overall memory requirements.

Launching MSFragger

Ensure that you have placed MSFragger.jar in your working directory and have modified the parameters file to reference your protein database. MSFragger generates auxiliary files during database search so it is critical that **MSFragger must have write access to the directories containing the protein database AND the MS/MS data files**.

Determine the amount of system memory available that you would like to make available to MSFragger. This will be specified by the Java maximum heap size parameter -Xmx (e.g. -Xmx3700M for 3700 MB or -Xmx8G for 8GB).

MSFragger takes the first argument as the input parameters file, followed by a list of one or more MS/MS data files.

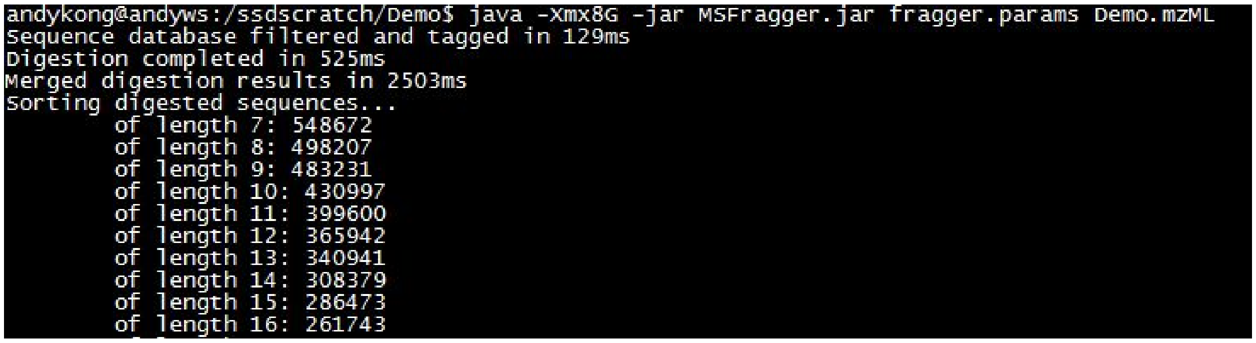
Examples:
java -Xmx8G MSFragger.jar fragger.params HeLa_run1.mzML HeLa_run2.mzML
java -Xmx8G MSFragger.jar fragger.params *.mzML

The **-Xmx flag is very important** to ensure that MSFragger has access to sufficient memory to efficiently perform the search as the default max heap setting in Java is ¼ of total system memory (which is insufficient for optimal performance). We recommend that you can allocate a minimum of 4G or 6G for standard tryptic digestions.

<u>Expected Behavior</u>

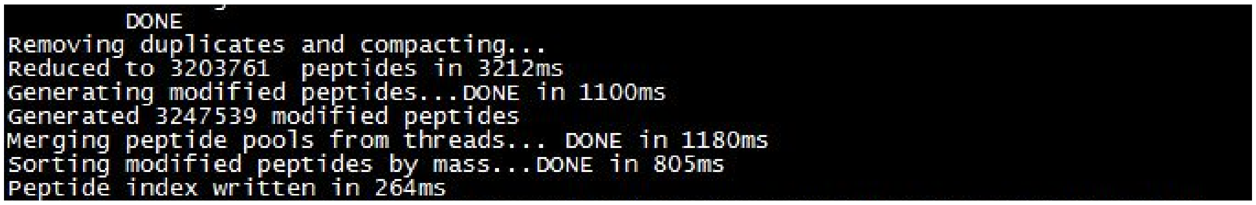
The first time running MSFragger on a new protein database or set of search parameters with a given database, it will first perform an in-silico digestion, create, and cache the peptide index (in .pepindex files adjacent to where the FASTA database is stored). These pepindex files can be safely removed at any time and should be removed to free up disk space when a set of search parameters is no longer used (MSFragger will automatically re-generate the index as needed).

The process begins with filtering and in-silico digestion subject to the digestion parameters.



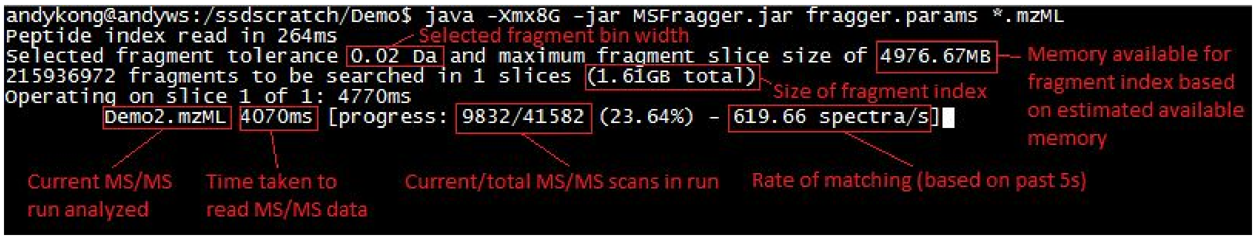
```
andykong@andyws:/ssdscratch/Demo$ java -Xmx8G -jar MSFragger.jar fragger.params Demo.mzML
Sequence database filtered and tagged in 129ms
Digestion completed in 525ms
Merged digestion results in 2503ms
Sorting digested sequences...
        of length 7: 548672
        of length 8: 498207
        of length 9: 483231
        of length 10: 430997
        of length 11: 399600
        of length 12: 365942
        of length 13: 340941
        of length 14: 308379
        of length 15: 286473
        of length 16: 261743
```

Followed by peptide sorting and de-duplication. The non-redundant set of peptides are then evaluated to generate the set of variably modified peptides (based on the specified variable modifications) which are then sorted by mass and stored.



```
        DONE
Removing duplicates and compacting...
Reduced to 3203761  peptides in 3212ms
Generating modified peptides...DONE in 1100ms
Generated 3247539 modified peptides
Merging peptide pools from threads... DONE in 1180ms
Sorting modified peptides by mass...DONE in 805ms
Peptide index written in 264ms
```

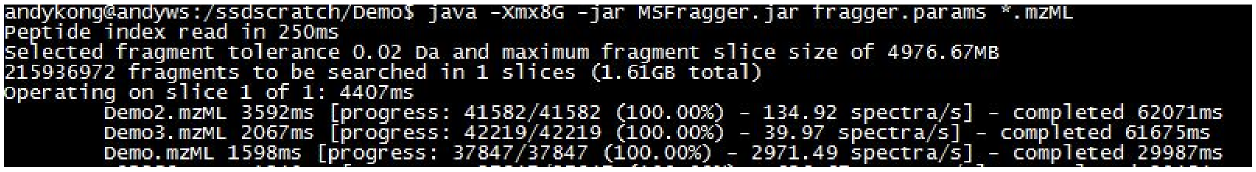
After peptide index generation is complete (or is read from disk in the below screenshot). MSFragger selects the fragment index bin width to use and estimates the memory available for fragment index storage based on the available memory (in this case, 8GB of memory was made available to the Java Virtual Machine, of which MSFragger estimates that 4976.67MB can be safely reserved for fragment index operations). It then computes the number of theoretical fragments to be generated for the entire index, the number of slices or iterations (in multi-pass searches when there is insufficient memory), and the total amount of memory represented by the entire fragment index. The fragment index is then generated, and a time is reported for the index generation time (at the end of each Operating on slice 1 of X: line, 4770 ms below). If the maximum fragment slice size is very small compared to your desired amount of system memory or the number of slices is unexpectedly high, double check that the -Xmx flag is correctly set.



```
andykong@andyws:/ssdscratch/Demo$ java -Xmx8G -jar MSFragger.jar fragger.params *.mzML
Peptide index read in 264ms
Selected fragment tolerance 0.02 Da and maximum fragment slice size of 4976.67MB
215936972 fragments to be searched in 1 slices (1.61GB total)
Operating on slice 1 of 1: 4770ms
        Demo2.mzML 4070ms [progress: 9832/41582 (23.64%) - 619.66 spectra/s]
```

Selected fragment bin width
Memory available for fragment index based on estimated available memory
Size of fragment index
Current MS/MS run analyzed
Time taken to read MS/MS data
Current/total MS/MS scans in run
Rate of matching (based on past 5s)

Search begins and the current file is reported, along with the time needed to read and pre-process the MS/MS data, along with current search progress.



```
andykong@andyws:/ssdscratch/Demo$ java -Xmx8G -jar MSFragger.jar fragger.params *.mzML
Peptide index read in 250ms
Selected fragment tolerance 0.02 Da and maximum fragment slice size of 4976.67MB
215936972 fragments to be searched in 1 slices (1.61GB total)
Operating on slice 1 of 1: 4407ms
        Demo2.mzML 3592ms [progress: 41582/41582 (100.00%) - 134.92 spectra/s] - completed 62071ms
        Demo3.mzML 2067ms [progress: 42219/42219 (100.00%) - 39.97 spectra/s] - completed 61675ms
        Demo.mzML 1598ms [progress: 37847/37847 (100.00%) - 2971.49 spectra/s] - completed 29987ms
```

At the completion of the search, a completed time is reported, and the results are written to disk in the same folder as the MS/MS data (if they are not in the same folder as your working directory). Note that there is a current bug that causes MSFragger to incorrectly display the average rate of matching at the conclusion of the run (although the total time can be divided by the total number of spectra to calculate this value).

<u>Output Files</u>

.fragtmp	In cases of fragment index fragmentation (in limited memory

	scenarios), MSFragger will iteratively load each MS/MS run and search loaded spectra against the current index slice before working on the next index slice. The partial search results are then stored in these .fragtmp files. In the event that MSFragger is terminated in the middle of a search, it will recover its partial results using these files. At the end of the last index slice, MSFragger will read all such .fragtmp files and generate an aggregated results file (identical to one that would be generated if it had the memory to search against all peptides in a single pass). These .fragtmp files are then automatically deleted. These can be safely removed if you no longer wish to continue an aborted search or if MSFragger somehow fails to remove them at the conclusion of a successful search. Location: Same directory as MS/MS files
.pepindex	MSFragger stores the computed peptide index in .pepindex files adjacent to the protein database files to remove the need to re-compute the index if search parameters are unchanged in subsequent runs. These .pepindex indices can be safely removed and MSFragger will re-compute the index again at runtime if needed. Location: Same directory as protein database
Results Files (eg. .pep.xml)	These are the pepXML or TSV output files containing the peptide identifications. The file extension is specified in the search parameters so specifying a .pep.xml extension with output_format = tsv will output .pep.xml files with TSV content. Location: Same directory as MS/MS files

Interpretation of Output

For pepXML outputs, these can be used for downstream processing using PeptideProphet in TPP directly. For viewing of results or conversion to other peptide identification result formats for use in other pipelines or tools that do not support pepXML, we recommend first converting to the mzIdentML format using the tool idconvert as part of the ProteoWizard package. The pepXML generated by MSFragger validates against v 1.18 of the pepXML schema and should be compatible with any downstream tools supporting the pepXML format.

The output fields of the TSV file produced by MSFragger are listed below:

ScanID
Precursor neutral mass (Da)
Retention time (minutes)
Precursor charge
Hit rank
Peptide Sequence
Upstream Amino Acid
Downstream Amino Acid
Protein
Matched fragment ions
Total possible number of matched theoretical fragment ions
Neutral mass of peptide (including any variable modifications) (Da)
Mass difference
Number of tryptic termini
Number of missed cleavages
Variable modifications detected
    (starts with M, separated by |, formated as position,mass)
Hyperscore
Next score
Intercept of expectation model (expectation in log space)
Slope of expectation model (expectation in log space)